

新人ゼミ課題

担当教員：早志 英朗 先生

提出者：片木 裕司

学籍番号：09B22022

提出年月日：2025 年 4 月 29 日

目 次

1 画像処理の基礎	2
1.0 python+OpenCV の環境構築	2
1.1 numpy を使った行列の四則演算	3
1.2 画像の表示, 縮小拡大, 回転, 二値化	3
1.2.1 画像の表示	4
1.2.2 画像の縮小拡大	4
1.2.3 画像の回転	4
1.2.4 画像の二値化	5
1.3 2枚の異なる画像の差分画像作成	6
1.4 画像の特徴量抽出と図示	7
1.4.1 ヒストグラム	7
1.4.2 ORB 特徴量抽出	7
1.4.3 SIFT 特徴量抽出	9
2 医用画像識別	10
2.1 環境構築	10
2.2 データセットの読み込み	10
2.3 データセットを成形	10
2.4 モデルの作成	10
2.5 学習	11
2.6 評価	11
2.7 混合行列	11
2.8 考察	12

本課題で使用したソースコードおよび関連ファイルは、GitHub 上に公開している。

- <https://github.com/UG-isLab/introduction.git>

以下に、プロジェクトのフォルダ構成を示す。

```
.  
├── NewcomersAssignment #課題の要件とデータが格納されているフォルダ  
├── image #課題 1 で使用した画像が格納されているフォルダ  
├── poetry.lock  
├── pyproject.toml  
├── report #本レポートが格納されているフォルダ  
├── result #課題 1, 課題 2 で生成した画像が格納されているフォルダ  
└── src  
    └── introduction  
        ├── __init__.py  
        ├── task1 #課題 1 のプログラムコードを格納するフォルダ  
        └── task2 #課題 2 のプログラムコードを格納するフォルダ  
└── tests #今回は使用しない
```

1 画像処理の基礎

1.0 python+OpenCV の環境構築

- OS : Ubuntu (WSL2 環境)
- エディタ : Visual Studio Code
- Python バージョン管理 : pyenv
- 仮想環境管理 : Poetry
 - opencv-python
 - opencv-contrib-python
 - matplotlib
 - numpy

まず、WSL2 を用いて Linux 環境を作成した。次に、pyenv を用いて python のインストールとバージョン管理を行った。python のバージョンは 3.12.9 を使用。poetry を用いて本課題用の仮想環境を立ち上げ、使用するライブラリを管理した。

1.1 numpy を使った行列の四則演算

```
a1 = np.array([[1, 2, 3], [4, 5, 6]])
a2 = np.array([[10, 20, 30], [40, 50, 60]])
a3 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

に対して

```
result1 : a1+a2
result2 : a2-a1
result3 : a2*a1
result4 : a2/a1
result5 : np.dot(a1, a3)
result6 : np.divide(a2, a1)
```

を行った結果は図 1 のようになった。

```
# introduction.py3.12ug-islab@UG-isLab:~/isLab/introduction$ /home/ug-islab/isLab/introduction/.venv/bin/python /home/ug-islab/isLab/introduction/n/src/introduction/task1/task1-1.py
result1 :
[[11 22 33]
 [44 55 66]]
result2 :
[[ 9 18 27]
 [36 45 54]]
result3 :
[[ 10 40 90]
 [160 250 360]]
result4 :
[[[10, 10, 10,]
 [10, 10, 10,]]
 [[10, 10, 10,]
 [10, 10, 10,]]]
result5:
[[ 38 44 50 56]
 [ 83 98 113 128]]
result6 :
[[[10, 10, 10,]
 [10, 10, 10,]]]
```

図 1: 四則演算結果

1.2 画像の表示、縮小拡大、回転、二値化

opencv を用いて画像の表示、縮小拡大、回転、二値化を行う。今回用いた写真は図 2 に示した。



図 2: 桜：昼

図 3: 桜：夜

1.2.1 画像の表示

opencv-pthon を用いて画像を表示する。画像の表示に用いる関数を以下に示す。

- cv2.imread(読み込む写真のパス)
写真を読み込むための関数。指定されたパスにある写真を numpy 配列として変数に格納する。
- cv2.imshow(' タイトル', 'imread で読み込んだ変数 (以降は img と表記)')
写真を表示するための関数。
- cv2.waitKey(0)
表示した写真を表示し続けるための関数。引数の数ミリ秒間写真を表示する。0 を指定した場合はキーが押されるまで表示し続ける。
- cv2.imwrite(' 写真を保存するパス', img)
読み込んだ写真を指定したパスに保存するための関数。

1.2.2 画像の縮小拡大

次に画像を拡大、縮小する。結果を図 4 と図 5 に示す。図 4 では固定サイズにより幅 100 ピクセル、高さ 300 ピクセルに拡大縮小したもので、図 5 では倍率指定により幅を 1.5 倍、高さを 2 倍に拡大している。画像の拡大縮小に用いた関数をいかに示す。

- cv2.resize(img,(幅, 高さ))
画像をピクセル単位で拡大縮小する。指定したタプルの画像が生成される。
- cv2.resize(img, None, fx=倍率, fy=倍率)
画像を倍率指定で拡大縮小する。None を指定することでピクセル単位の拡大縮小でないことを明記。

1.2.3 画像の回転

次に画像を回転する。結果を図 6 に示す。また、回転するのに使用したコードを以下に示す。

```
1 /height, width = img.shape[:2]
2 center = (width // 2, height // 2)
3 matrix = cv2.getRotationMatrix2D(center, 45, 1.0)
4 rotated_img = cv2.warpAffine(img, matrix, (width, height))
```

まず、1 行目では shpae() メソッドを用いることで画像の幅と高さを取り出し、2 行目で写真の中央の座標を定めている。3 行目では getRotationMatrix2D() 関数を用いて画像を回転+拡大縮小するためのアフィン行列を作成する。引数は ('回転の中心', '回転の角度', 'スケール') を指定する。最後に 4 行目で warpAffine() 関数を用いて画像にアフィン変換を適用する。引数は (img, '適用するアフィン行列', '出力する画像のサイズ') を指定する。つまり今回は等倍で画像の真ん中を中心として時計回りに 45 度回転した画像を生成している。



図 4: 幅 100 ピクセル、高さ
300 ピクセル



図 5: 幅 1.5 倍、高さ 2 倍

1.2.4 画像の二値化

次に画像を二値化する。結果を図 7に示す。二値化とは画像の各ピクセルを黒 (0) または白 (255) のいづれかにすることを指す。二値化に用いた関数を以下に示す。

- cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
カラー画像をグレースケールに変換するための関数。画像の二値化は 1 チャンネルの画像に
対して行うものであるため、グレースケール化することで各ピクセルを輝度で表している。
- cv2.threshold('グレースケール化した画像', '閾値', '白として使う値', cv2.THRESH_BINARY)
グレースケール化した画像を閾値を値をもとに閾値よりも低い値を黒 (0), 高い値を白 (255)
にする。cv2.THRESH_BINARY により白黒変換を適用する。また、戻り値は実際に使われ
た閾値と出力画像の二つが出力されるが今回は後者のみを使用する。



図 6: 画像の回転



図 7: 画像の二値化

1.3 2枚の異なる画像の差分画像作成

この章では2枚の異なる画像の差分画像を作成する。比較する二枚の画像は図2と図3で、この二つは同じ場所から撮影した桜であり、一枚目昼、二枚目は夜に撮影している。また、差分画像の結果を図8に示す。差分画像を作成するのに用いた関数を以下に示す。

- cv2.absdiff(img1, img2)

引数に二つの写真を指定することで差分画像を作成することができる。差があるところは白くなり、差がないところは黒くなる。

二つの写真を比べる前処理として二つの写真をグレースケール化することにより輝度の違いが強調され、違いが分かりやすくなるようにした。

差分画像を見てみると木の幹の部分はほとんど変わらず、ライトがあったっておらず昼間と比べると暗くなっている桜の木の上のほうが白くなっていることがわかる。また昼と夜で空の明るさも変わっていることがわかる。



図 8: 差分画像

1.4 画像の特徴量抽出と図示

1.4.1 ヒストグラム

図 2 と図 3 に対してヒストグラムを作成する。結果を図 9 と 10 に示す。また、ヒストグラムに作成した関数を以下に示す。

- cv2.calcHist(img, ' 使用するチャンネル', None, ' ヒストグラムのビン数', ' 画素の範囲')
画像のヒストグラムを計算するための関数。使用するチャンネルは赤、青、緑の 3 チャンネル。第 3 引数は画像の一部のみを対象にする場合に使用する。ヒストグラムのビン数と画素の範囲は通常 [256] と [0,256]
- plt.plot(' ヒストグラムの値', color=' 線の色') matplotlib の折れ線グラフを書くための関数。
第一引数に calcHist() で取得した値、第二引数に線の色を指定する。
- plt.savefig(" 画像を保存するためパス")
作成したヒストグラムを保存するための関数
- plt.show()
作成したヒストグラムを表示するための関数

このヒストグラムは横軸が 0-255 の画素値、縦軸がその値をとるピクセルの総数を赤、緑、青それぞれのチャンネルで表している。

ヒストグラムを見ると、昼に撮影した桜は右側に、夜に撮影した桜は左側に山がある。横軸が画素値であるため右側に行くほど白色に近くなり明るい色になっていることから、昼の桜の写真は全体的に明るく、夜の桜の写真は全体的に暗いことがヒストグラムからわかる。実際の写真を見てもその結果は明らかであり、それぞれの写真の特徴を正確にとらえられていることがわかる。

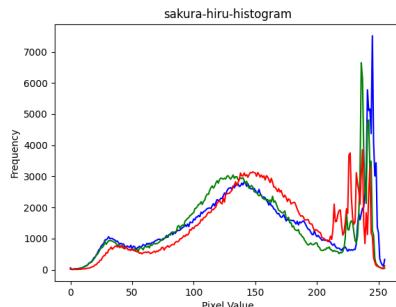


図 9: 昼の桜のヒストグラム

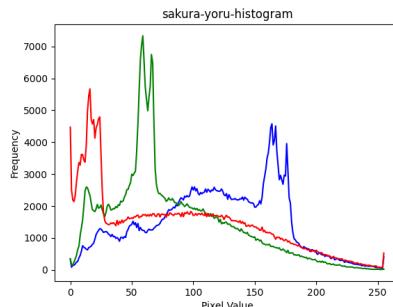


図 10: 夜の桜のヒストグラム

1.4.2 ORB 特徴量抽出

次に ORB 特徴量抽出を用いて昼の桜と夜の桜の特徴点を抽出して比較してみる。図 11 と 図 12 では各写真の特徴点を緑色の点で示した。使用した関数を以下で説明する。

- orb.detectAndCompute(' グレースケールにした写真', ' マスク')
画像を抽出して、その特徴量を計算する関数。写真全体の特徴量を得るために第二引数に None を指定。

- `cv2.drawKeypoints(img, 'detectAndCompute()' で得た変数', '出力先', 'color', '描画オプション')`

検出した特徴点を画像の上に目で見えるように描画する関数。第二引数に `None` を指定することで新しい画像が生成される。`color` では `(0, 255, 0)` とすることで緑を指定。

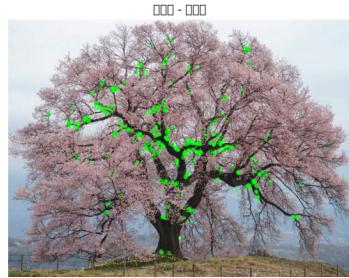


図 11: 昼の桜の ORB 特徴量抽出

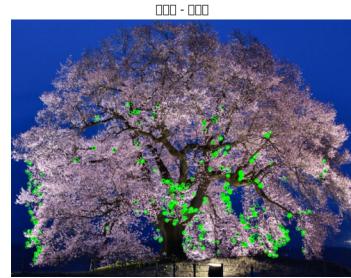


図 12: 夜の桜の ORB 特徴量抽出

図 13では二つの写真の特徴点をマッチングして、よく似た場所を 20 本線で可視化している。以下に使用した関数を示す。

- `cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)`
特徴量のマッチング器を作る関数。第一引数でマッチングにハミング距離を使うことを指定。第二引数では $[A \rightarrow B]$ と $[B \rightarrow A]$ の両方で最も近いものを採用する。これによりより確実なマッチングだけが残る。
- `bf.match(des1, des2)`
`detectAndCompute()` で取得した特徴ベクトルを用いてハミング距離が短いペアを探す関数。
- `sorted(matches, key=lambda x: x.distance)`
ラムダ関数を用いて距離が近い順にマッチングをソートする。これによりよくマッチした順番に並び替えることができる

図 13では二つの写真の全く異なる点同士がマッチしており、そこまで精度がよくないようと考えられる。

より高い精度のマッチングを行うために SIFT 特徴量抽出を試してみる。

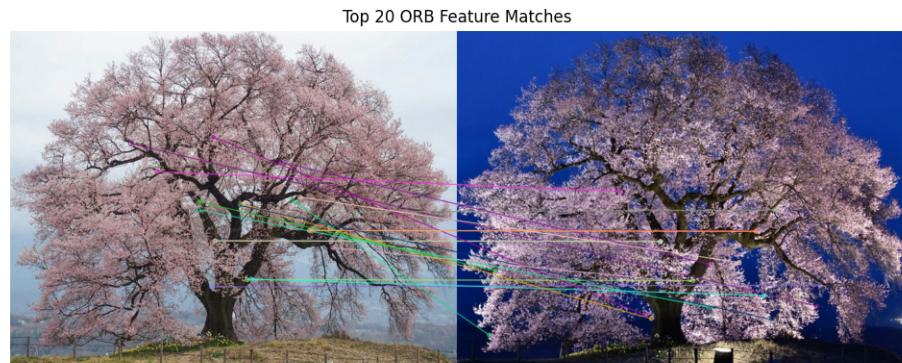


図 13: ORB 特微量マッチ

1.4.3 SIFT 特微量抽出

図 14にSIFT 特微量抽出によって抽出された特徴点マッチング結果を示した。以下に SIFT 特微量抽出で用いた関数を示す。

- `cv2.SIFT_create()`
SIFT オブジェクトを作成する関数。
- `sift.detectAndCompute(img, 'マスク')`
画像から特徴点を検出して特徴点の周囲から特微量を作成する関数。返り値は特徴点のリストと各特徴点に対応する 128 次元のベクトル。
- `cv2.BFMatcher()`
特微量マッチング器を作成する関数。
- `bf.knnMatch('detectAndCompute()' で取得した 128 次元のベクトル', 'detectAndCompute()' で取得した 128 次元のベクトル', k=2)`
knn マッチングを実施する関数。k=2 を指定することで 1,2 番目に近い 2 つの特微量を探す。これによりローの比率テストを行うために必要。
- ローの比較テスト
1 番目と 2 番目の距離を比較して 1 番目が 2 番目の 0.75 倍より小さい場合この二つが「十分違う」とみなしてマッチングした特徴点として使用する。これによりあいまいなマッチを排除できる。

図 14では ORB の時と比べてマッチングしている座標が一致しており精度よく特徴点マッチングができているとわかる。



図 14: SIFT 特微量マッチ

2 医用画像識別

この章では MRI 画像の腫瘍の有無による識別を行った。

2.1 環境構築

- 実行環境 : Google Colab
- ライブラリ : TensorFlow
- モデル構成 : CNN

2.2 データセットの読み込み

まずデータセットを読み込む。Google Drive に圧縮されたデータを置き、Google Colab でマウントして読み込んだ。次に tarfile を用いて圧縮されたデータを解凍する。学習データを train、検証データを valid、テストデータを test に格納した。

2.3 データセットを成形

次に keras.utils.image_dataset_from_directory() を用いてデータセットを作成する。この関数はサブディレクトリ名をクラスとして自動的にラベリングしてくれる。その際、画像サイズを 64, 64、バッチサイズを 32、シャッフルを True にする。それぞれのデータを train_ds, valid_ds, test_ds とした。最後にすべての画像データを 255 で割ることでデータを 0-1 の範囲に正規化する。さらに、tf.data.AUTOTUNE を用いて次のバッチデータを先に読み込むことで学習の効率を上げることができる。

2.4 モデルの作成

次に CNN で学習するためのモデルを作成する。以下に CNN モデルの構成を示す。

- 畳み込み層+最大プーリング層を 3 セット
- Flatten 層により 2 次元特徴マップを 1 次元に変換

- 全結合層を 2 層
- 活性化関数には中間層で ReLU 関数、出力層で softmax 関数を使用した

さらにモデルのコンパイルを定義する。モデルのコンパイルには以下の設定を用いた。

- 最適化アルゴリズム：アダム
- 損失関数：クロスエントロピー
- 評価指標：正解率

2.5 学習

ここでは学習データと検証データを用いてモデルの学習を行う。
まずアーリーストッピングを設定する。これは過学習しないために行うもので検証データの損失関数が 3 エポック連続で改善されなかった場合に終了するよう設定した。また restor_best_weights を True にすることで過去の最もよかったモデルを保存するようにした。

次に fit() 関数を用いて学習を行う。ここでは引数として学習データ、検証データ、エポック数（ここでは 20 エポック）、先ほど設定したアーリーストッピングを指定した。

2.6 評価

次に evaluate() を用いてモデルの評価を行う。引数にテストデータを指定している。結果は正確率が 0.9243 と高い精度を示した。

2.7 混合行列

最後に評価したテストデータに対して混合行列を出力した。図 15 にその結果示す。

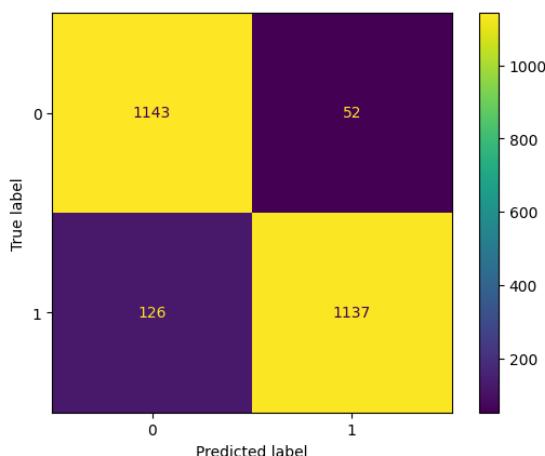


図 15: 混合行列

2.8 考察

結果から正確率は 92%を超えていたため高い精度であることがわかる。また、混合行列を確認してみても腫瘍のあり、なしのどちらでも正解数が 1143 と 1137 で偏りなく正解できていることがわかる。一方で誤分類のうち腫瘍ありをなしと分類した数が 126 とやや多い結果となった。医療の分野では、腫瘍があるのになしと誤判定されるのは医療の分野ではなるべく避けたいものであるためこれを改善する必要があると考える。

これを改善する策として、損失関数に重みをつけることで腫瘍ありの誤分類に対してより大きなペナルティを与える要することが考えられる。